

000001-990002.60

10 A

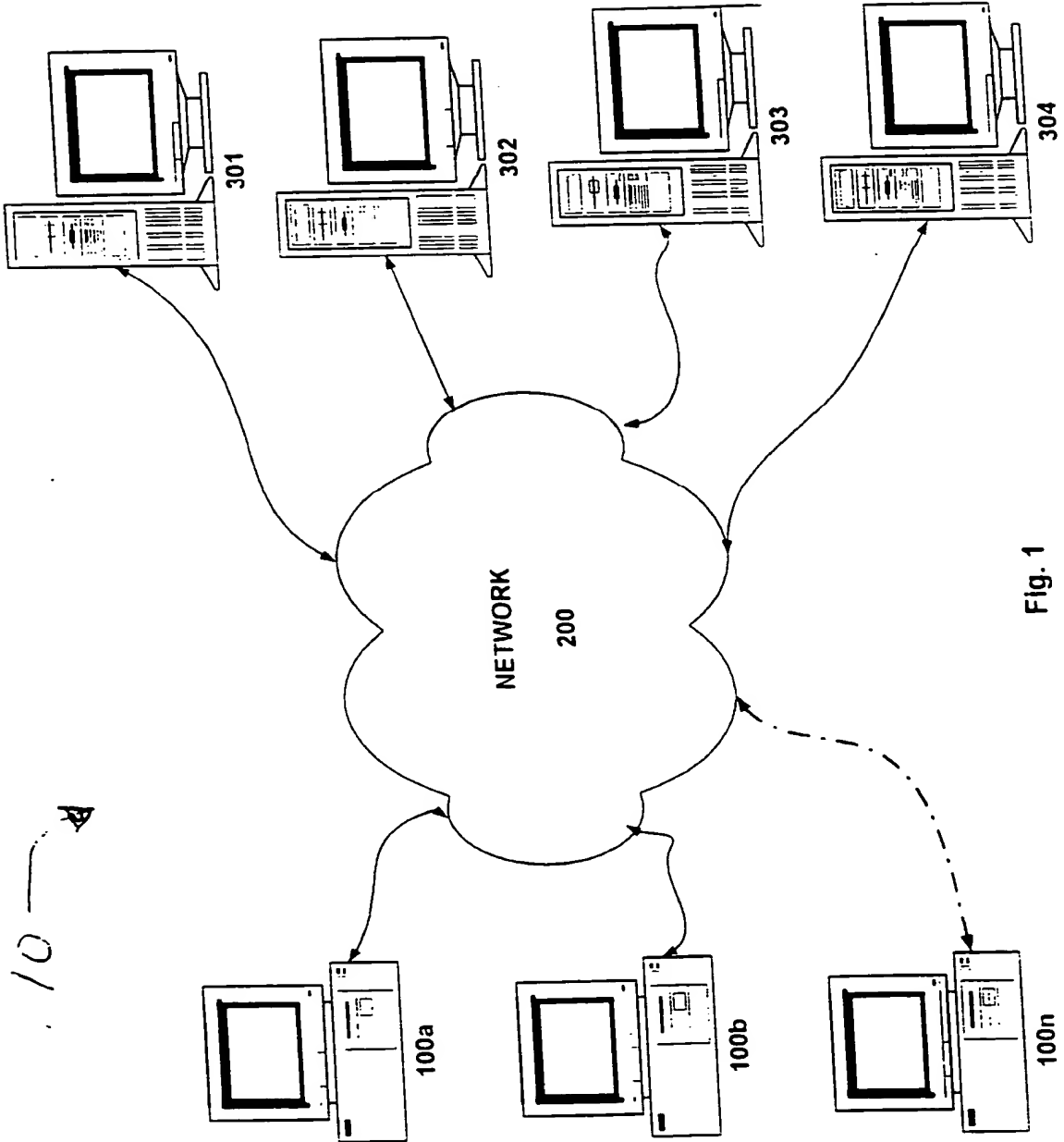


Fig. 1

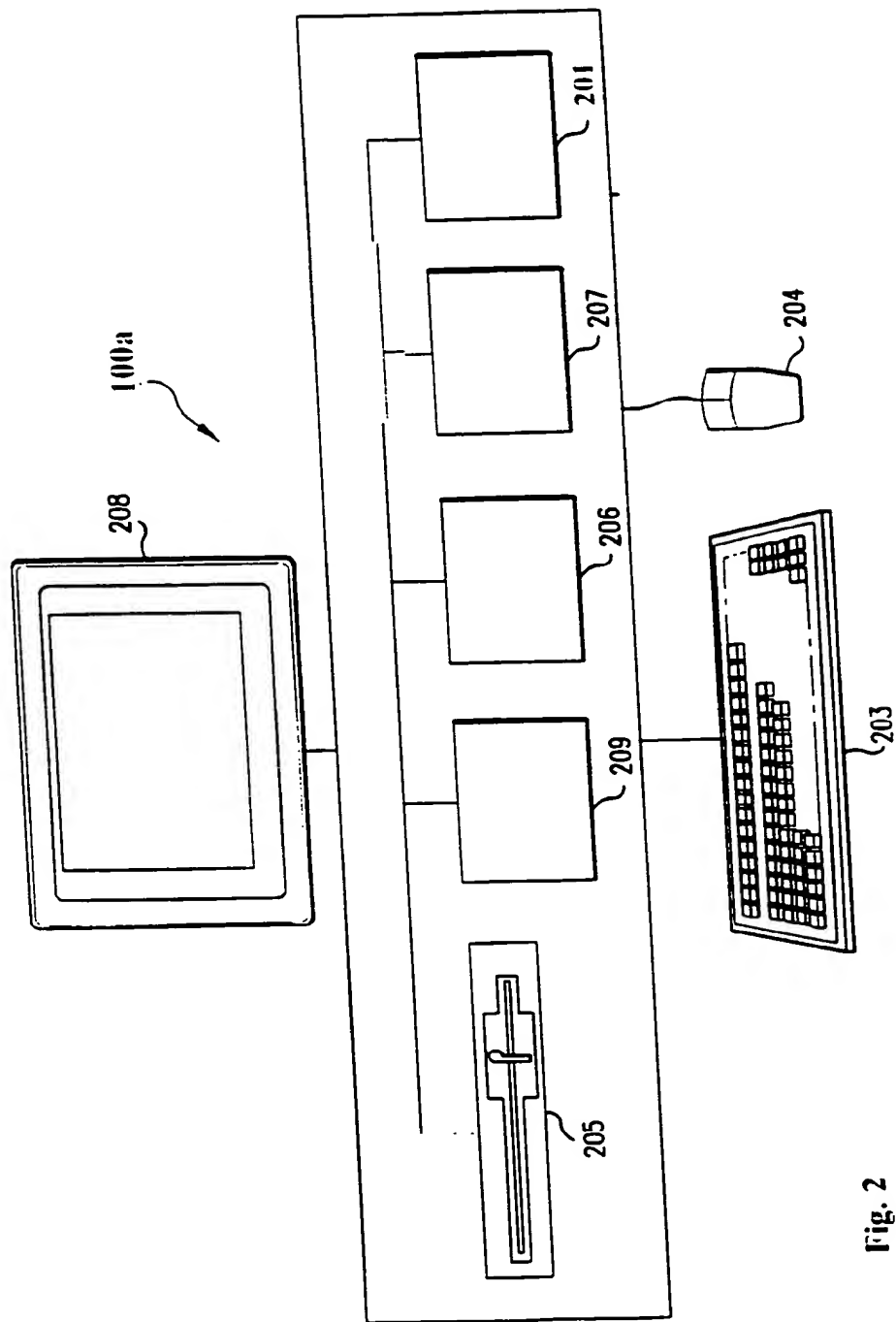


Fig. 2



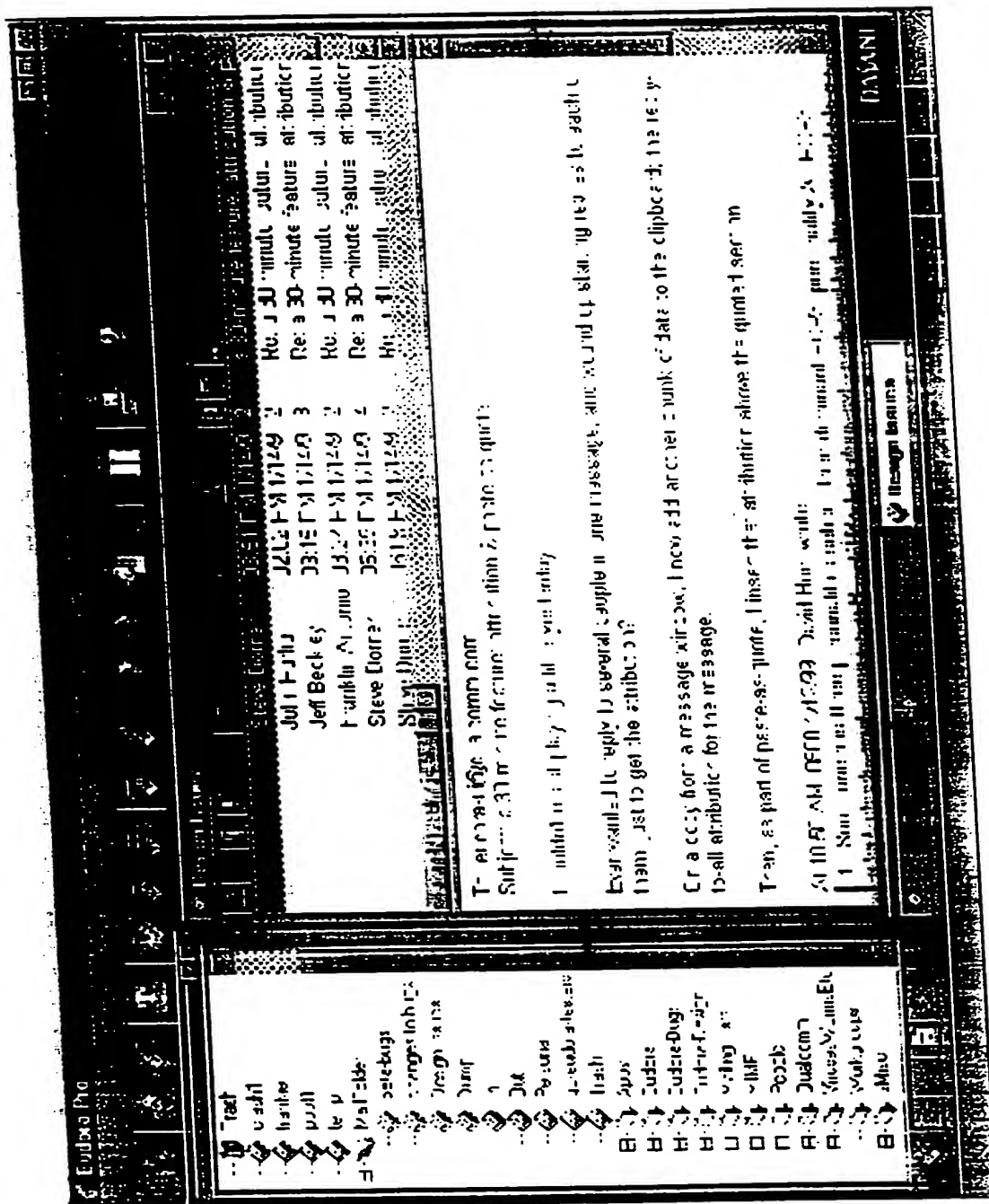
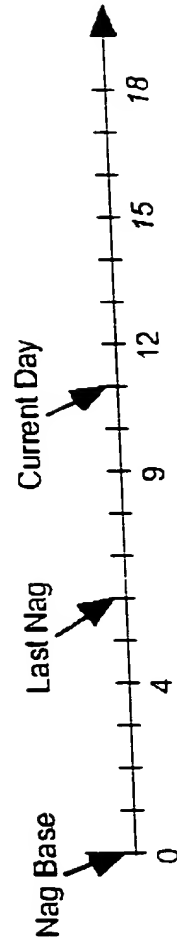
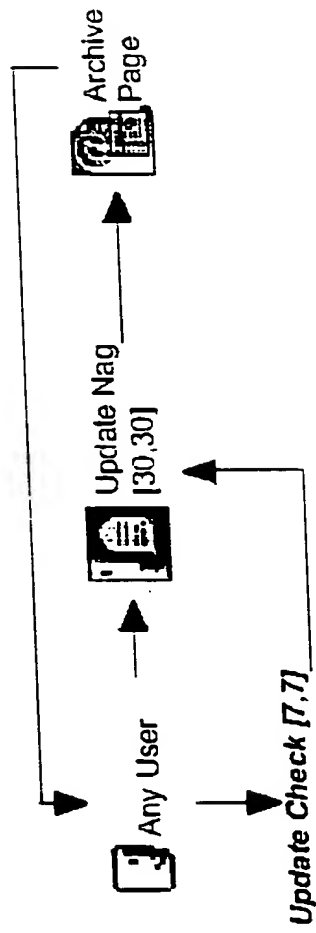
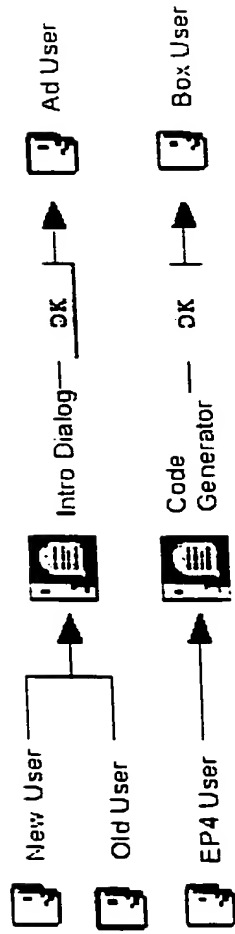


Fig. 3B'



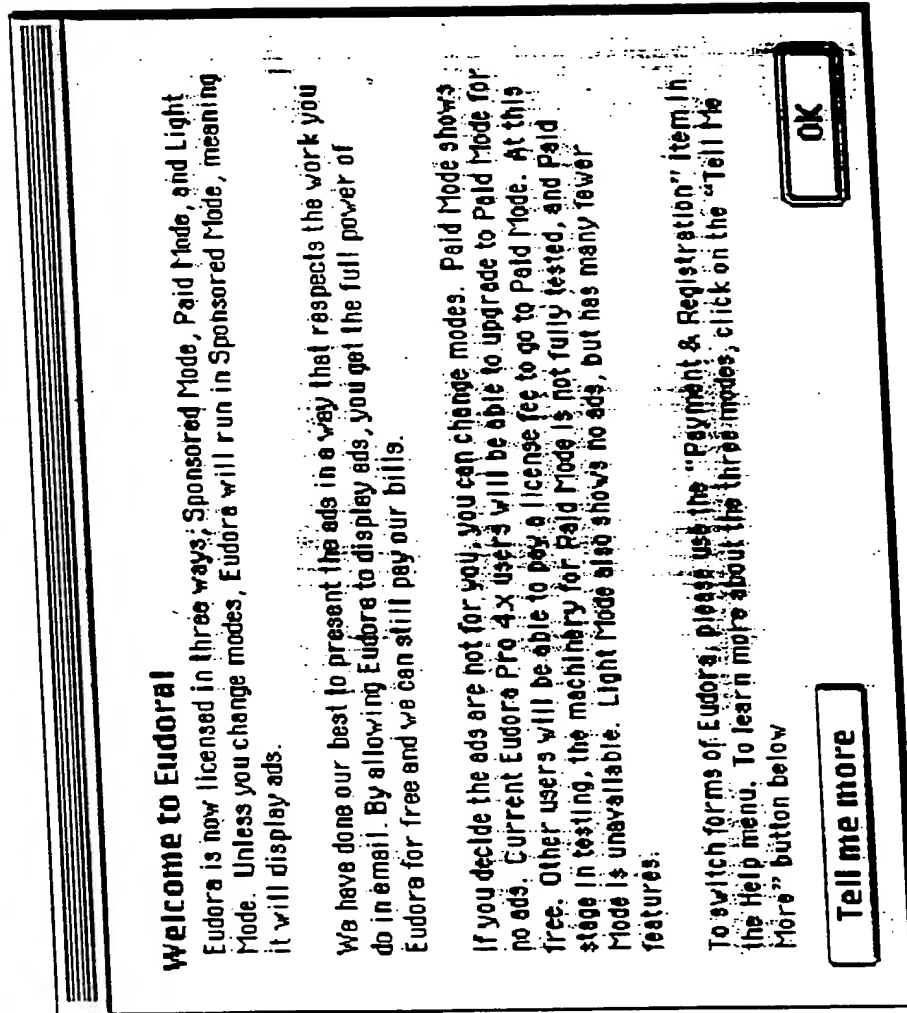


Fig. 4B

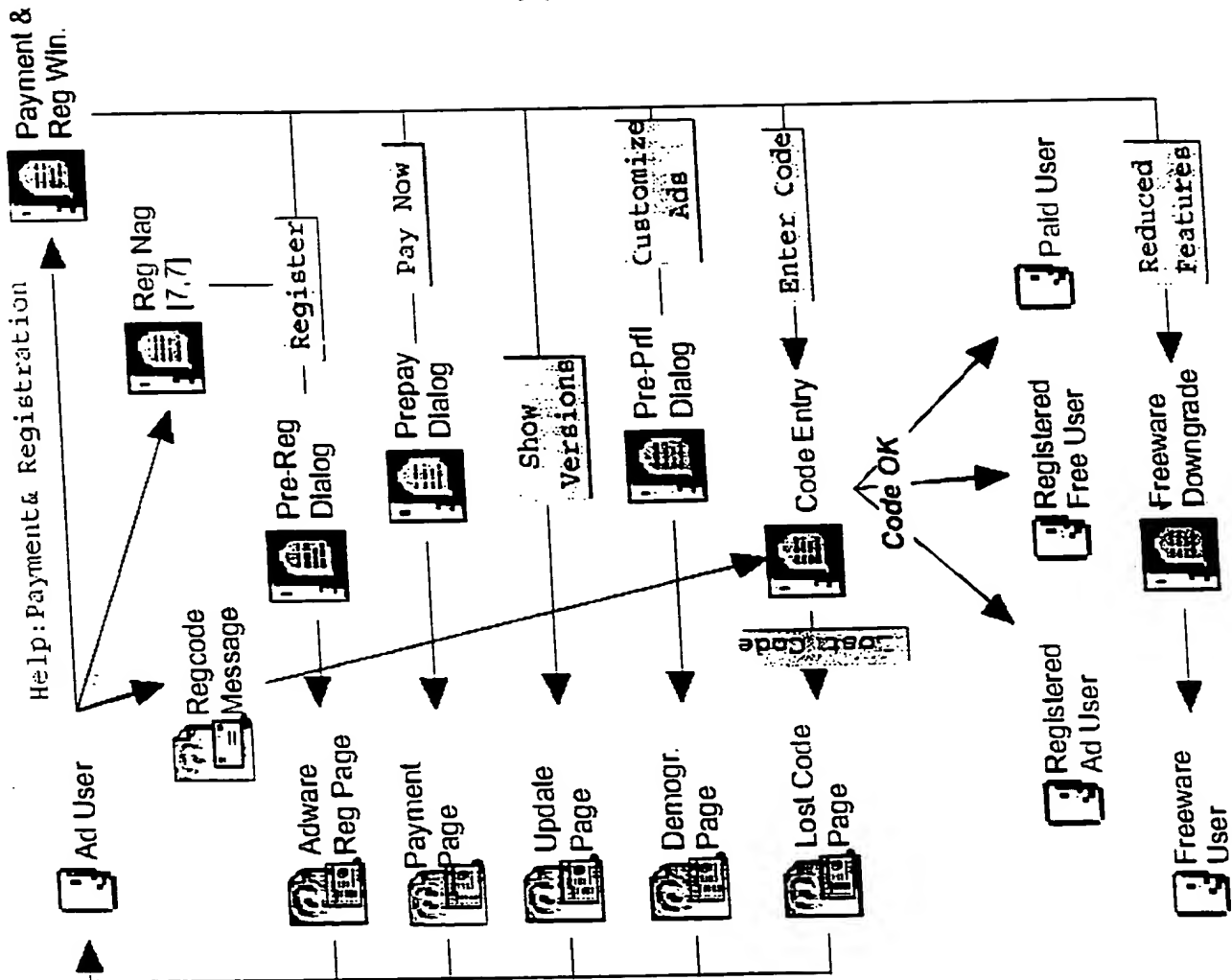




Fig. 5A

Payment & Registration


Which Eudora is right for you?



Sponsored Mode  
(free, with ads)




Paid Mode  
(costs money, no ads)




Light Mode  
(free, fewer features)


Keeping Current



Register with Us




Customize the  
Ads You See



Find the Latest  
Update to Eudora

Your Registration Information



Change Your  
Registration

<no registration name>  
<no registration code>

Take me to Eudora's home page

Fig. 5B



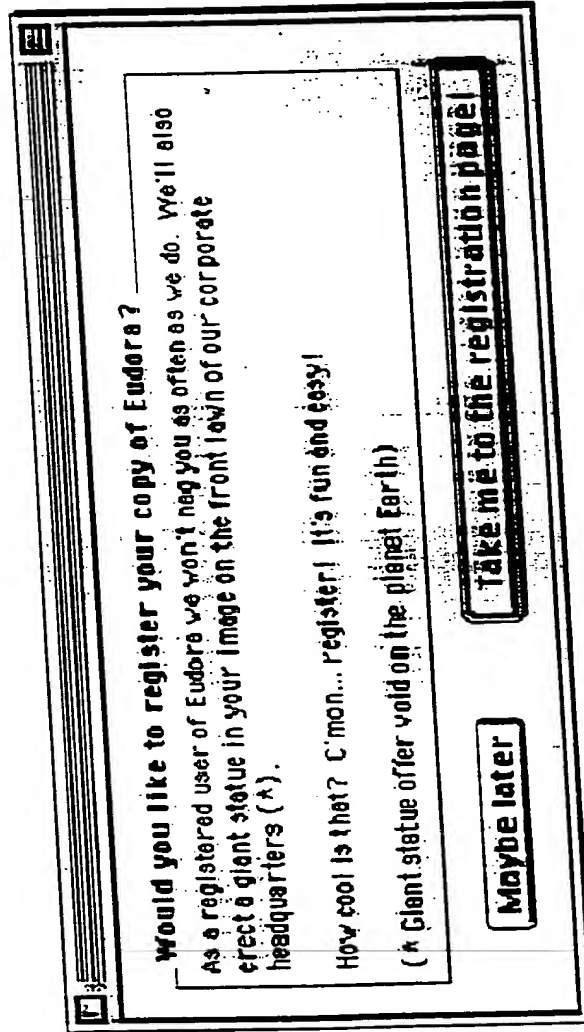


Fig. 5C

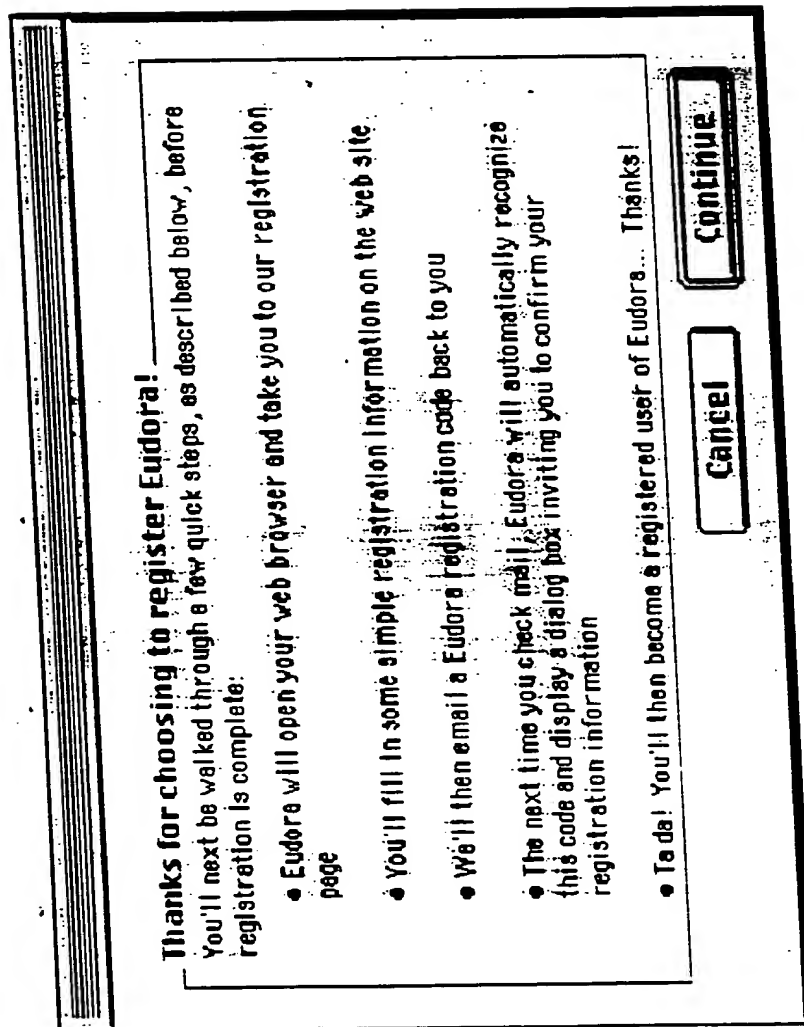


Fig. 5D

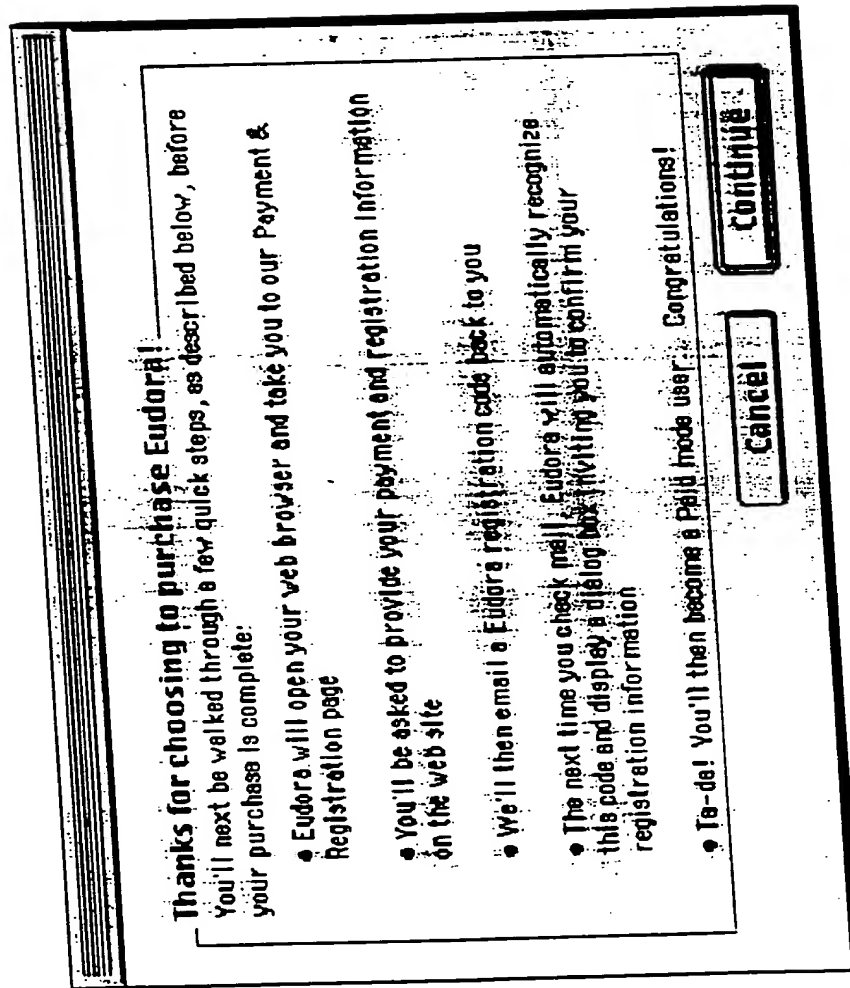


Fig. 5E

Thank you for your registration!

To complete your registration, please enter the name you ordered and your registration code below.

The exact name you registered under:

First Name: John

Last Name: Manyjars

Your registration code:

48925-89A2-B1149

I lost the Code

Cancel

OK

**Fig. 5F**

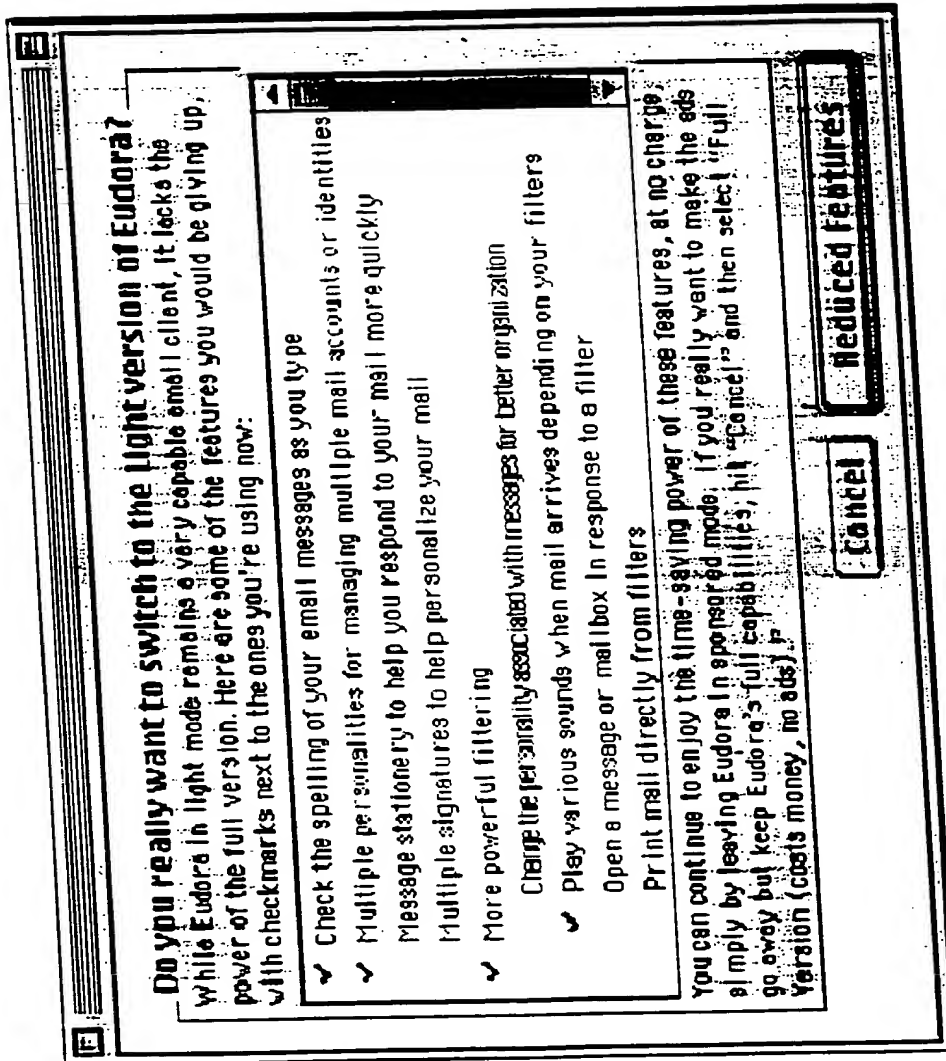


Fig. 5G

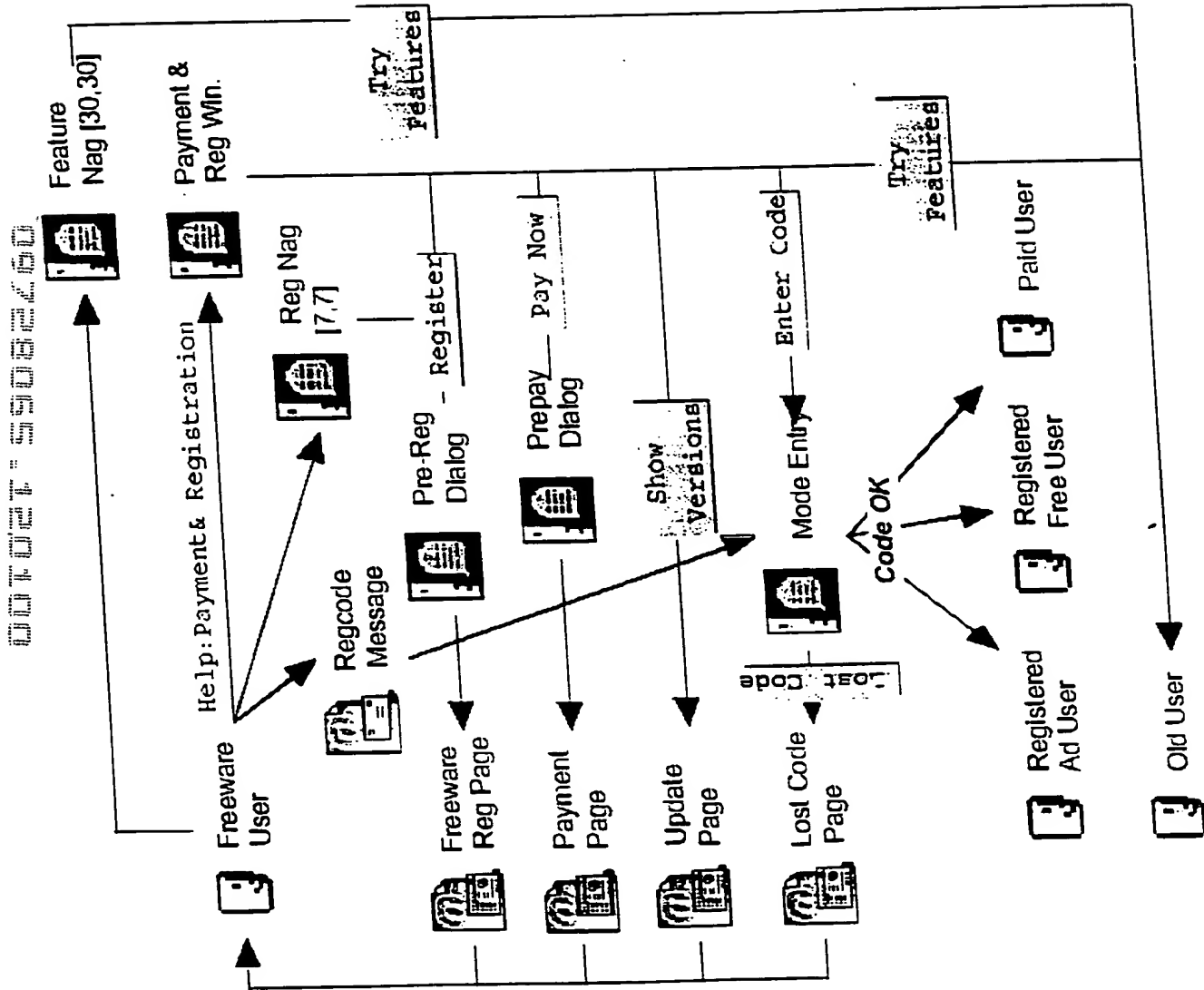


Fig. 6A

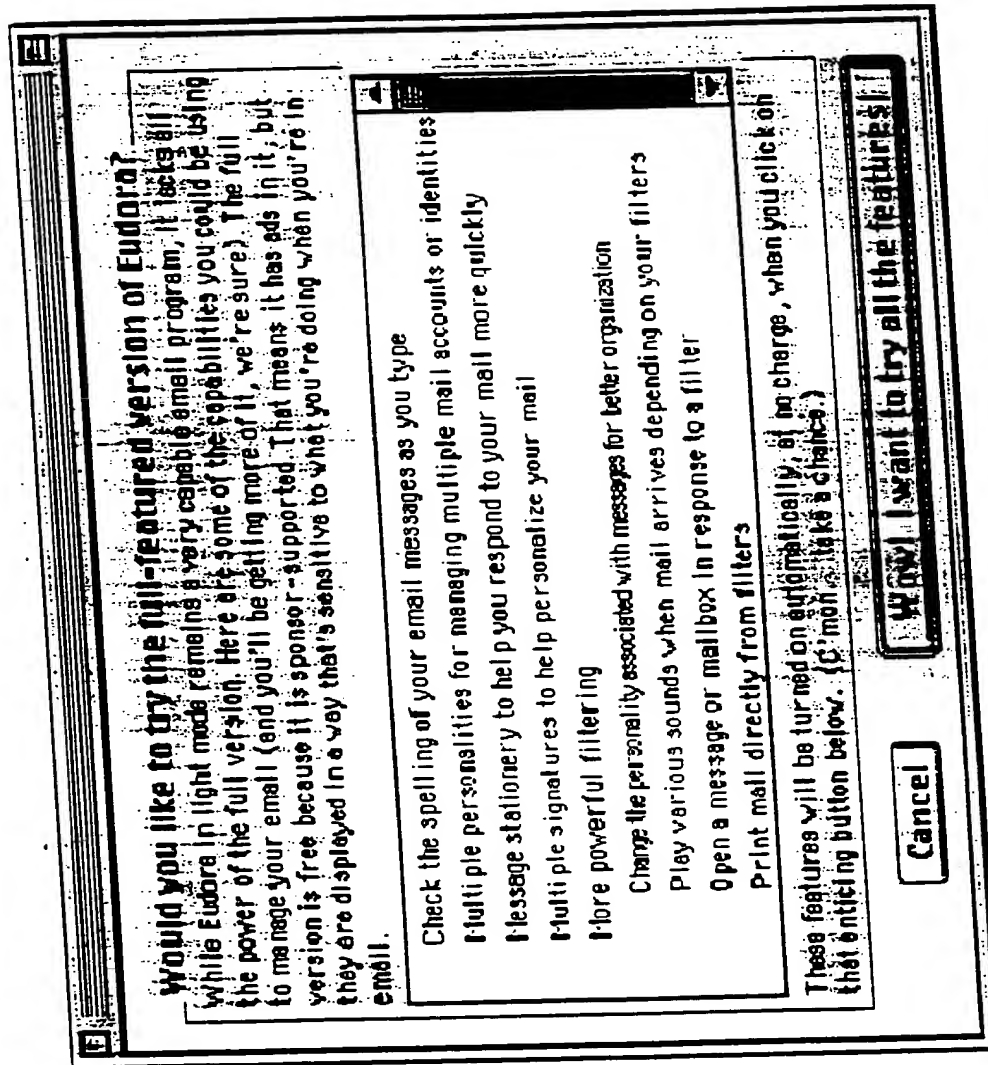
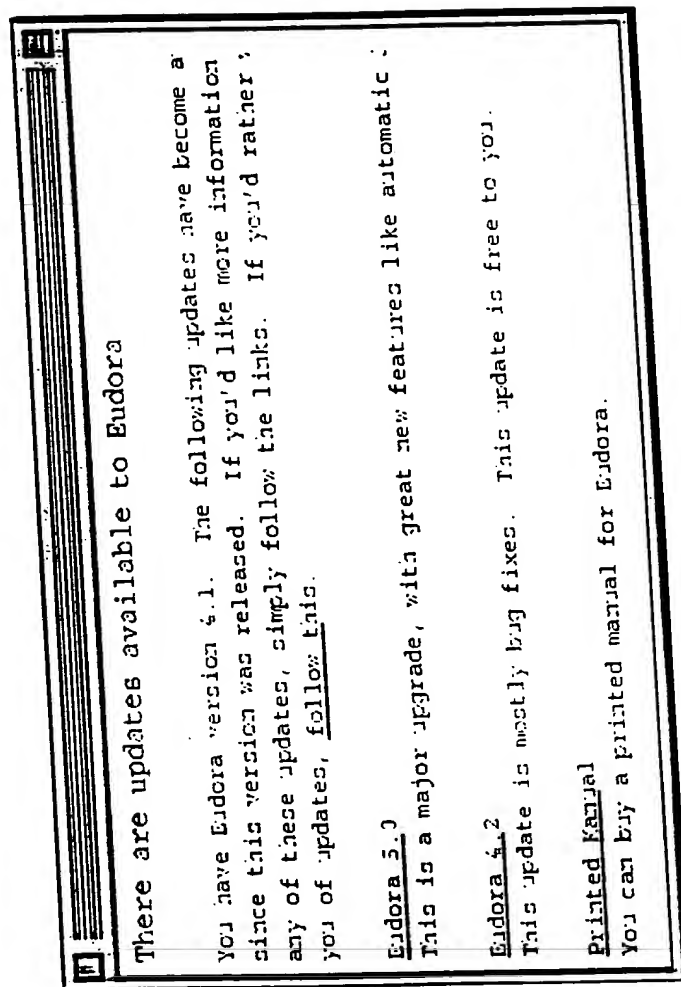


Fig. 6B



**Fig. 7B**



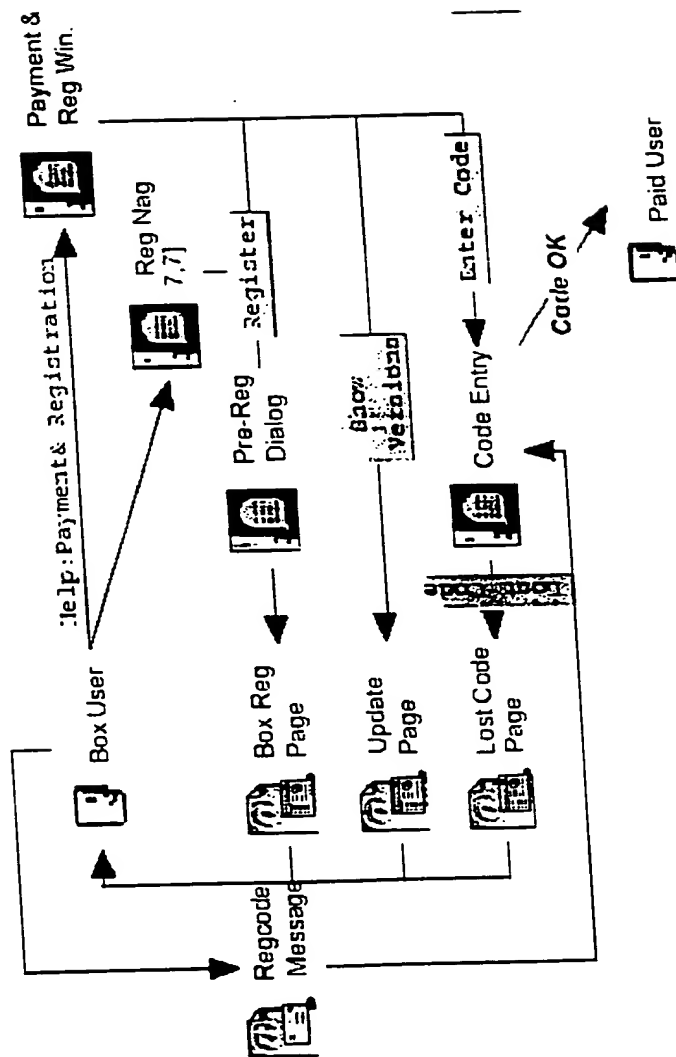


Fig. 8

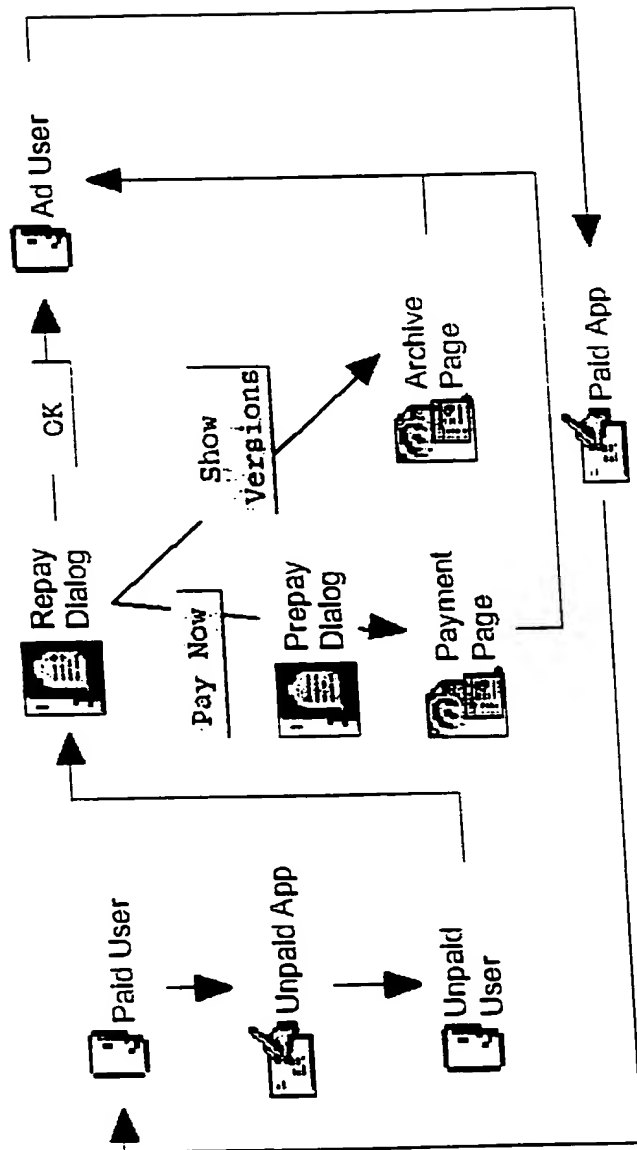


Fig. 9

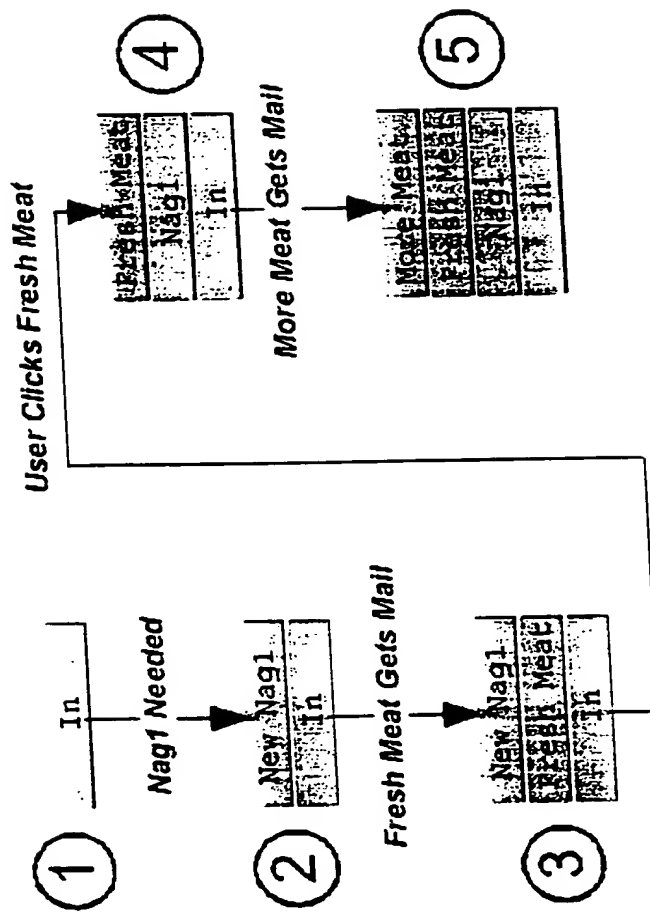
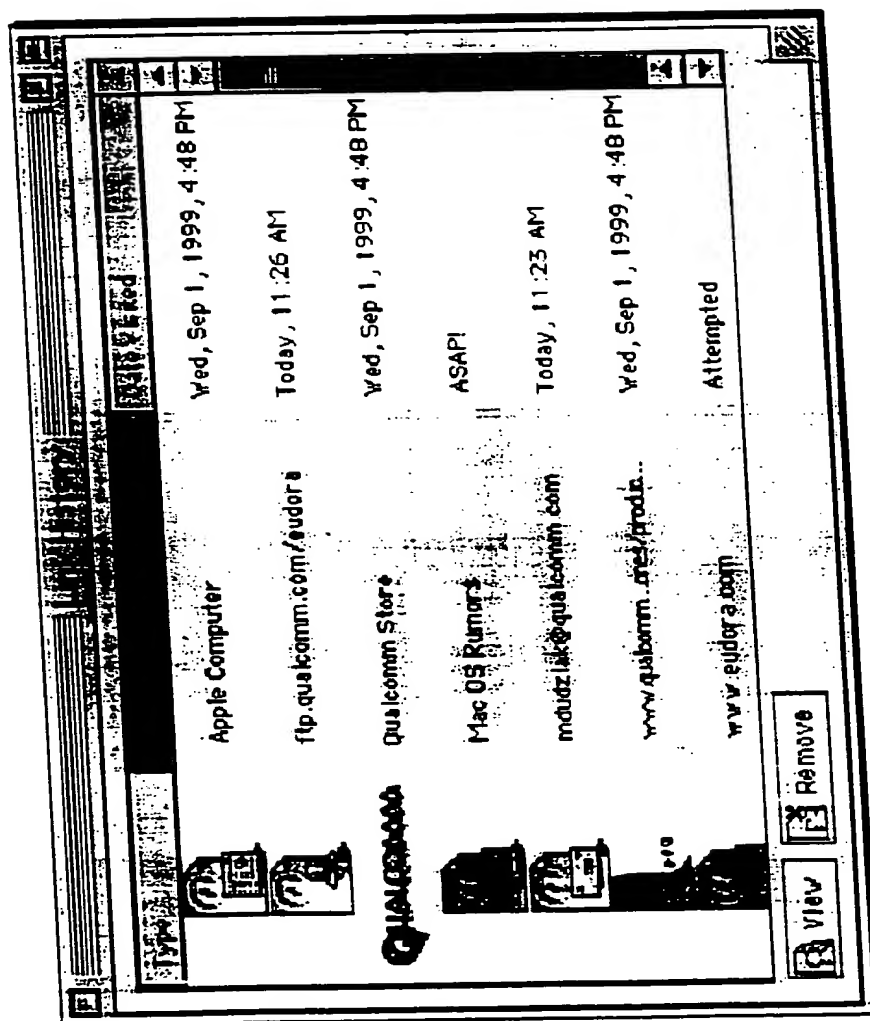


Fig. 10

[illegible]

**Fig. 12A**

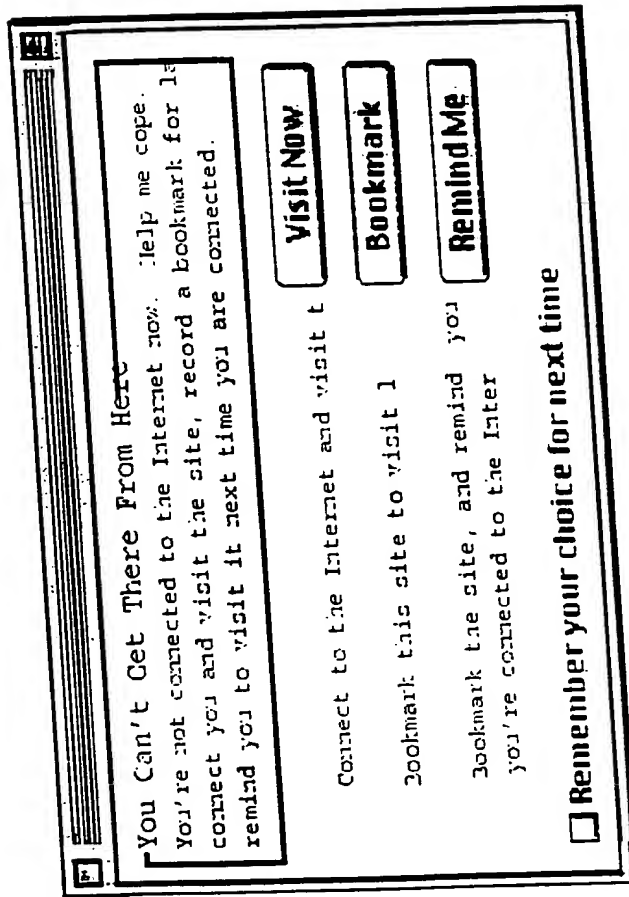


Fig. 12B



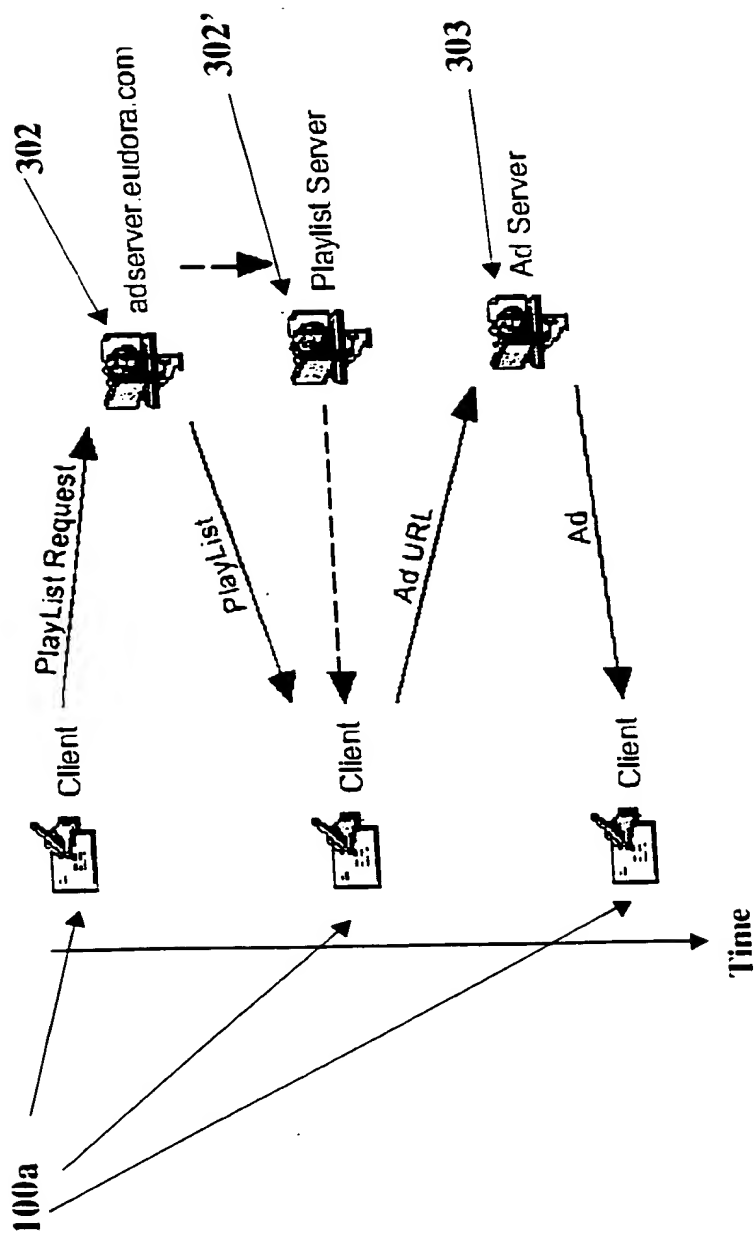


Fig. 14





```

////////////////////
// We must perform certain tasks when the calendar day
changes.
CheckForNewDay
(if ( the calendar day has changed )
{
// Perform housekeeping tasks on the ad currently showing
Do StopShowingCurrentAd
// Runout ads are charged for a full showFor if they've been
shown
// at all on a given day. Charge any runout ads if they've
been
// shown at all.
for runout ads
{
if ( ad.thisShowTime > 0 )
{
ad.totalTimesShown += ad.showFor
ad.thisShowTime = 0
}
}
// Now, reset the counters for all ads to reflect the fact
that
// a new day has dawned.
for all ads
{
ad.numberShownToday = 0
}
// Record yesterday's facetime
// Might not literally be yesterday, be sure to use
// whatever day the app was last run on
set old current day's facetime to totalFaceTimeToday
// and reset our global regular ad facetime counter
adFaceTimeToday = 0
totalFaceTimeToday = 0
// if we were in a block, back out
set block to all playlists
}
}
// end CheckForNewDay

```

**Fig. 15B**

007027-59082250

```
////////////////////////////////////
// This function shows a runout ad, and if it
// can't find one, goes to a rerun
ShowARunout
{
for runout ads
{
// has the ad been flushed?
if ( ad.flushed )
try next ad
// are we done showing this runout today?
if ( ad.numberShownToday > ad.dayMax )
try next ad // this one's used up for the day
// are we done showing this runout for ever and ever?
if ( ad.shownFor > ad.showForMax )
try next runout ad // this one's used up forever
// are we between the ad's start and end dates?
if ( ad.startDate < the current date < ad.endDate )
try next runout ad
// the ad is not supposed to run today
// do we actually HAVE the ad?
if ( ad has not been downloaded )
{
ask for ad to be downloaded
try next ad
}
// ok, we believe we should show this runout
// we are now in runout state
Do ShowAnAd
return
}
// if we haven't found a runout ad, we will go to "rerun"
state
Do ShowARerun
}
// end ShowARunout
```

Fig. 15C

00755055-1000  
007027-55052760

```
////////////////////////////////////  
// Rerun state. Look for a regular ad to rerun  
ShowARerun  
{  
  for regular ads [ in current block ]  
  {  
    // has the ad been flushed?  
    if ( ad.flushed )  
      try next ad  
    // is this ad recent enough to rerun?  
    if ( ad.lastShownDate is older than returnInterval )  
      try next ad  
    // this one is too old to rerun  
    // if in block, show ads only if it's their "turn"  
    if ( ad.numberShownToday >= blockGoal )  
      try next ad // need to find a friend in this block  
    // are we between the ad's start and end dates?  
    if ( ad.startDate < the current date < ad.endDate )  
      try next ad  
    // the ad is not supposed to run today  
    // do we actually HAVE the ad?  
    if ( ad has not been downloaded )  
    {  
      ask for ad to be downloaded  
      try next ad  
    }  
    // ok, at this point we can show this ad, but because  
    // we're in rerun, we don't keep the books  
    Do ShowAnAd  
    return  
  }  
  // if we get here, we have no ads to show. Punt.  
  return  
}  
// end ShowARerun
```

Fig. 15D

```

////////////////////////////////////
// Show a regular ad
ShowARegularAd
{
for regular ads [ in current block ]
{
// has the ad been flushed?
if ( ad.flushed )
try next ad
// are we done showing this ad today?
if ( ad.numberShownToday > ad.dayMax )
try next ad // this one's used up for the day
// if in block, show ads only if it's their "turn"
if ( ad.numberShownToday >= blockGoal )
try next ad // need to find a friend in this block
// are we done showing this ad for ever and ever?
if ( ad.shownFor > ad.showForMax )
try next ad // this one's used up forever
// are we between the ad's start and end dates?
if ( ad.startDate < the current date < ad.endDate )
try next ad
// the ad is not supposed to run today
// do we actually HAVE the ad?
if ( ad has not been downloaded )
{
ask for ad to be downloaded
try next ad
}
// ok, we believe we should show this ad
// we are now in regular state
Do ShowAnAd
return
}
// If we get here, we have failed to find a regular
// ad. Go to runout
Do ShowARunout
}
// end ShowARegularAd

```

Fig. 15E

```

////////////////////////////////////
// Perform necessary housekeeping when we're taking
// down an ad
AdEndBookkeeping
{
// In rerun state, we don't do any bookkeeping
if ( in RerunState )
return
// Account for at most ad.showFor seconds, provided
// we've shown the ad for at least ad.showFor seconds
// Note that this means we don't charge for time beyond
// ad.showFor seconds, which is important
if ( ad.thisShowTime >= ad.showFor )
{
ad.numberShownToday += ad.showFor
ad.shownFor++
// we do NOT reset thisShowTime here, we do it in
// AdStartBookkeeping. It actually doesn't matter where
// we do it, provided we are careful NOT to do it for
// runout ads.
}
}
// end AdEndBookkeeping

```

007027-59032250

Fig. 15F

007027-59032260

```
////////////////////////////////////
// Show an ad, including bookkeeping and block handling
ShowAnAd
{
  // If the ad is in a block, notice that
  if ( it's in a "block" playlist )
  {
    if ( not currently in a block )
    {
      find ad in block with minimum numbersShown
      make that our ad
      set blockGoal to minimum numbersShown+1
    }
    set current block to this playlist
  }
  // now do bookkeeping
  Do AdStartBookkeeping
  // and actually show it
  Do DisplayThatAd
}
```

Fig. 15G

```

////////////////////////////////////
// Perform housekeeping when we put up an ad
AdStartBookkeeping
{
// In rerun state, we don't do any bookkeeping
if ( in RerunState )
return
// For regular ads
if ( it's a regular ad )
{
ad.thisShowTime = 0
ad.lastShownDate = now
}
}
// end AdStartBookkeeping

```

00728065-120100

Fig. 15H

Persistent Ads	
PlayList Request	faceTime Used to determine how much advertising to send to client faceTimeLeft Not used
playList Response ClientInfo	reqInterval Relatively large: one or more days flush Used. Single playlist completely specifies list of ads client should have
PlayList Response Scheduling Parameters	showForMax Not used

Fig. 16A

Short-Lived Ads	
PlayList Request	faceTime Not used faceTimeLeft Used to determine how many ads client should receive
playList Response ClientInfo	reqInterval Not used. Instead, client requests new playlist whenever ads "run low". flush Not used
PlayList Response Scheduling Parameters	showForMax Used to determine how long an ad runs

Fig. 16B



000001-59032260

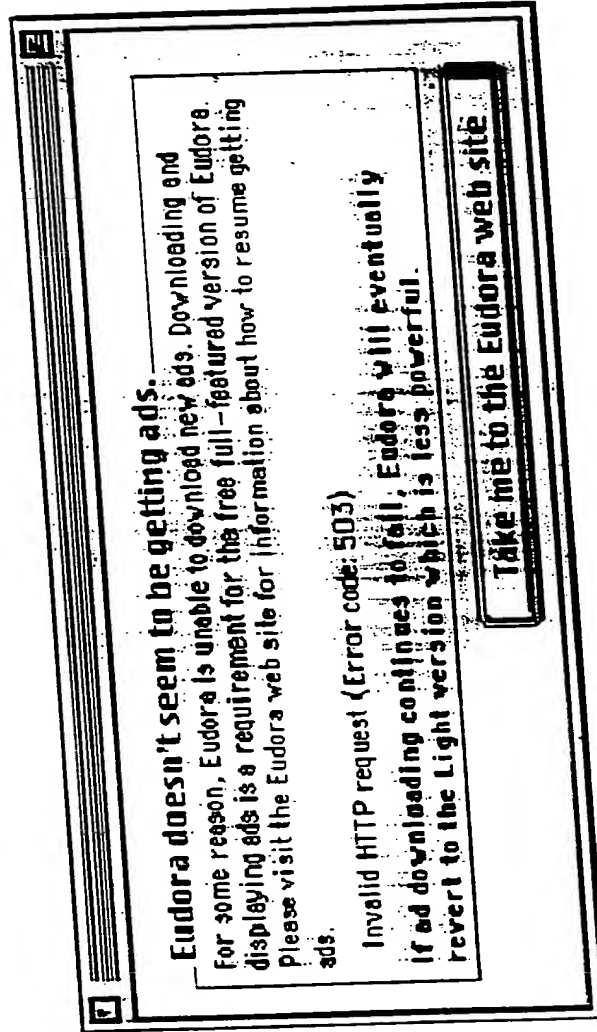


Fig. 17A

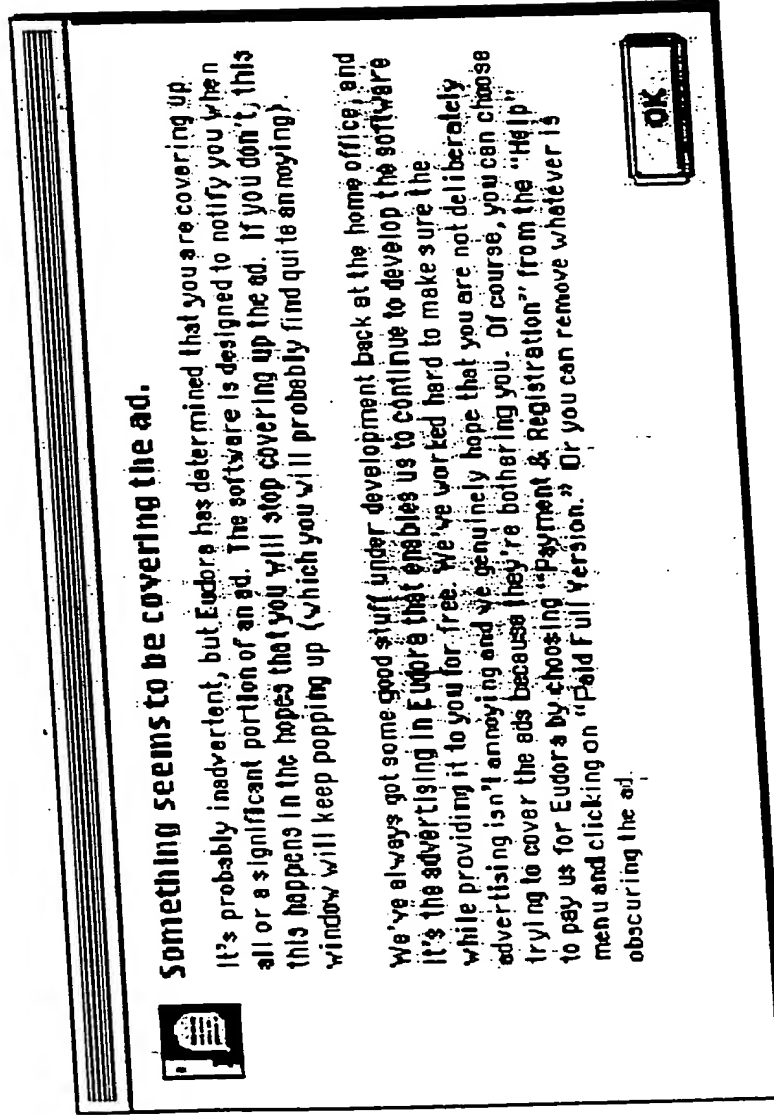


Fig. 17B

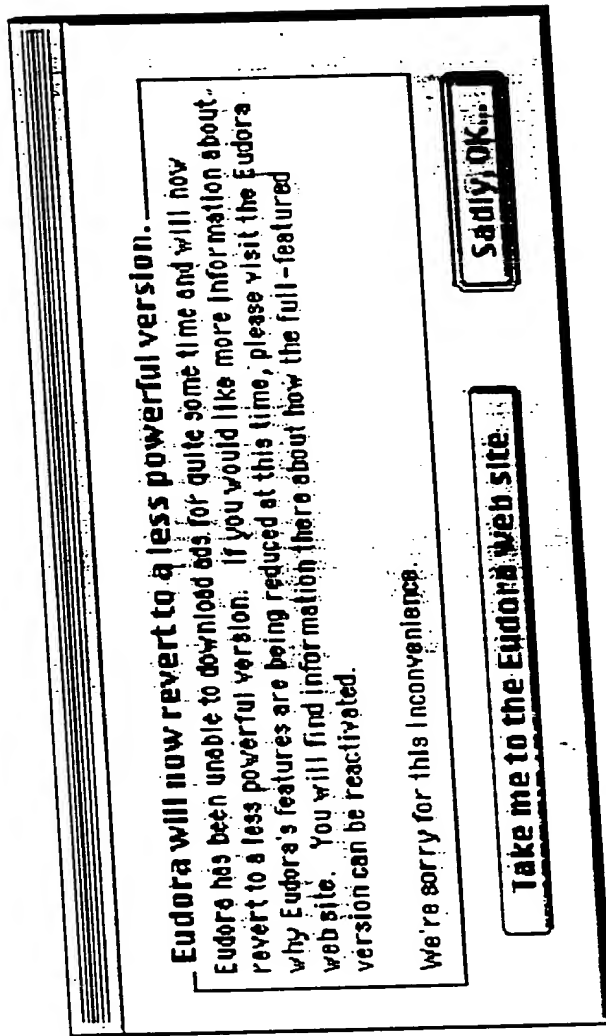


Fig. 17C





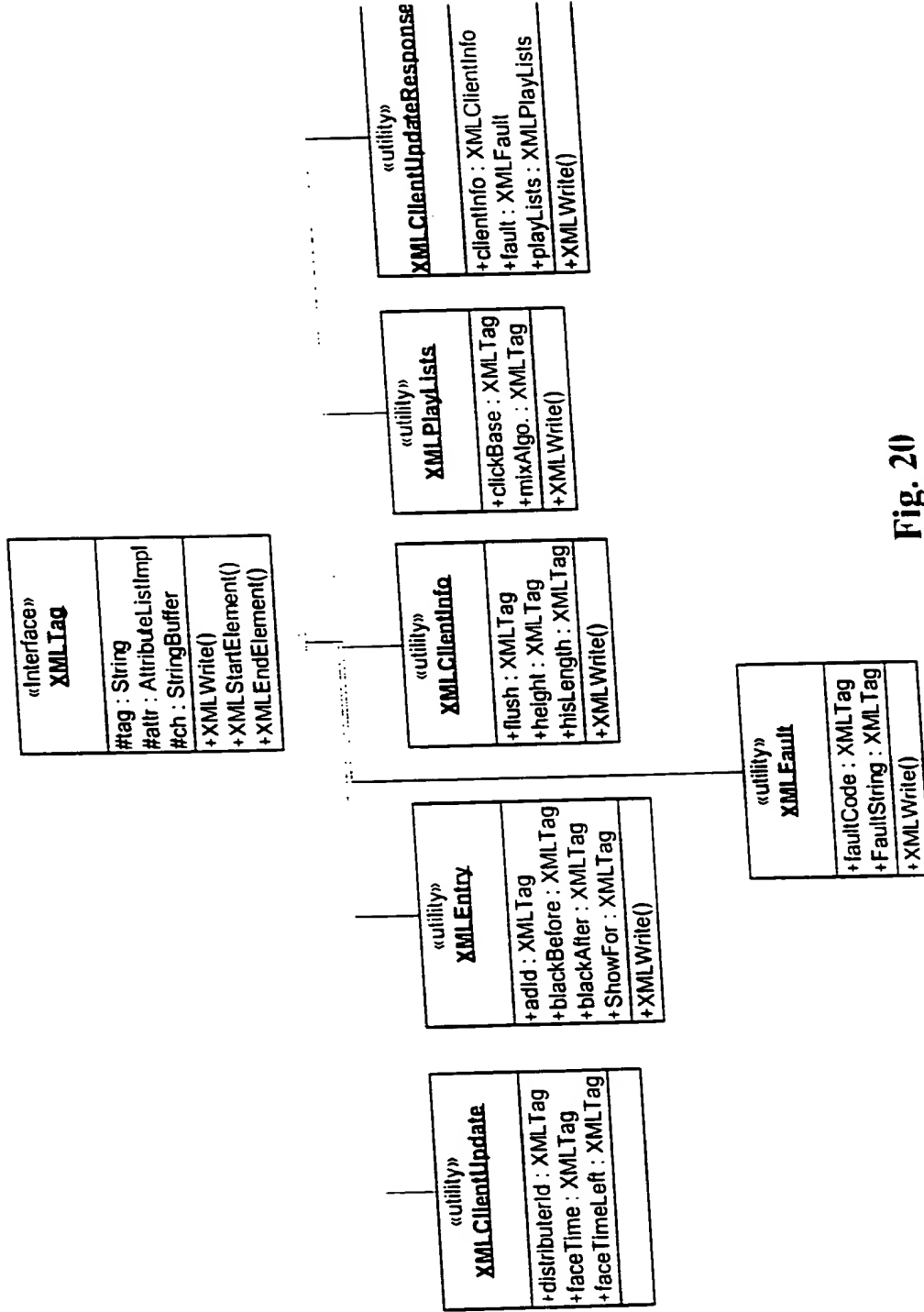


Fig. 20

8 The list of available ads advantageously can be built from the following query:

```
ads = dbCon.prepareStatement("SELECT * FROM ads WHERE StartDate <= today AND endDate >= today + 30 AND
AdType = 'p' AND AdStatus = 'A' AND ImpressionsServed < Impressions ORDER BY ImpressionsServed
ASC");

run out ads = dbCon.prepareStatement("SELECT * FROM ads WHERE StartDate <= today AND endDate >= today +
30 AND AdType = 'R' AND AdStatus = 'A' AND ImpressionsServed < Impressions ORDER BY ImpressionsServed
ASC);
```

8 The time required to deliver the ads advantageously can be calculated in the following manner.

face time left for today [seconds] = faceTime[today] - faceTimeUsedToday

(Comment: Face time left for today is the number of seconds the servlet can use to deliver special ads today.)

(Comment: Face time left for today is the number of seconds the servlet can use to deliver special ads today.)

predict face time [seconds] = SUM(faceTime[tomorrow], faceTime[tomorrow + 1], ... faceTime[tomorrow + reqInterval])

(Comment: Predict face time is the number of seconds the servlet predicts the user is going to have.)

goal show time left [seconds] = predict face time - faceTimeLeft

(Comment: Goal show time left is the number of seconds that the software provider needs to fill with ads.)

Fig. 21A

```

3 Targeting
  while (face time left for today ) {
    if ad is not in the history {
      select ad [according to target = today ]
      face time left for today -= ad.showFor
    }
    next ad
  }

  while (Goal show time left ) {
    if ad is not in the history {
      select ad [according to target]
      goal show time left -= ad.showFor
    }
    next ad
  }

```

Default values:

- reqInterval = 1 day.
- faceTime = 30 minutes
- faceTimeQuota is ?
- histLength = 31 days

Fig. 21B



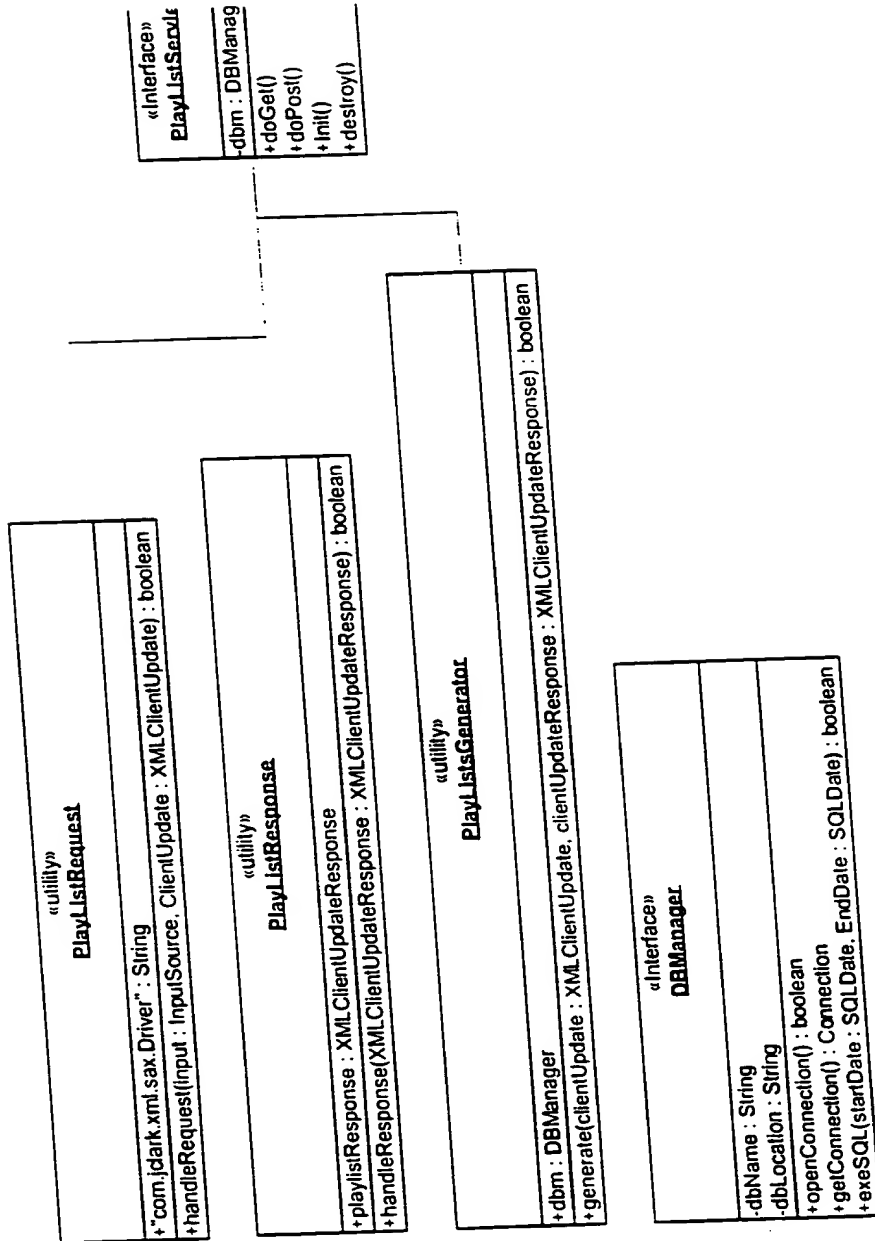


Fig. 22

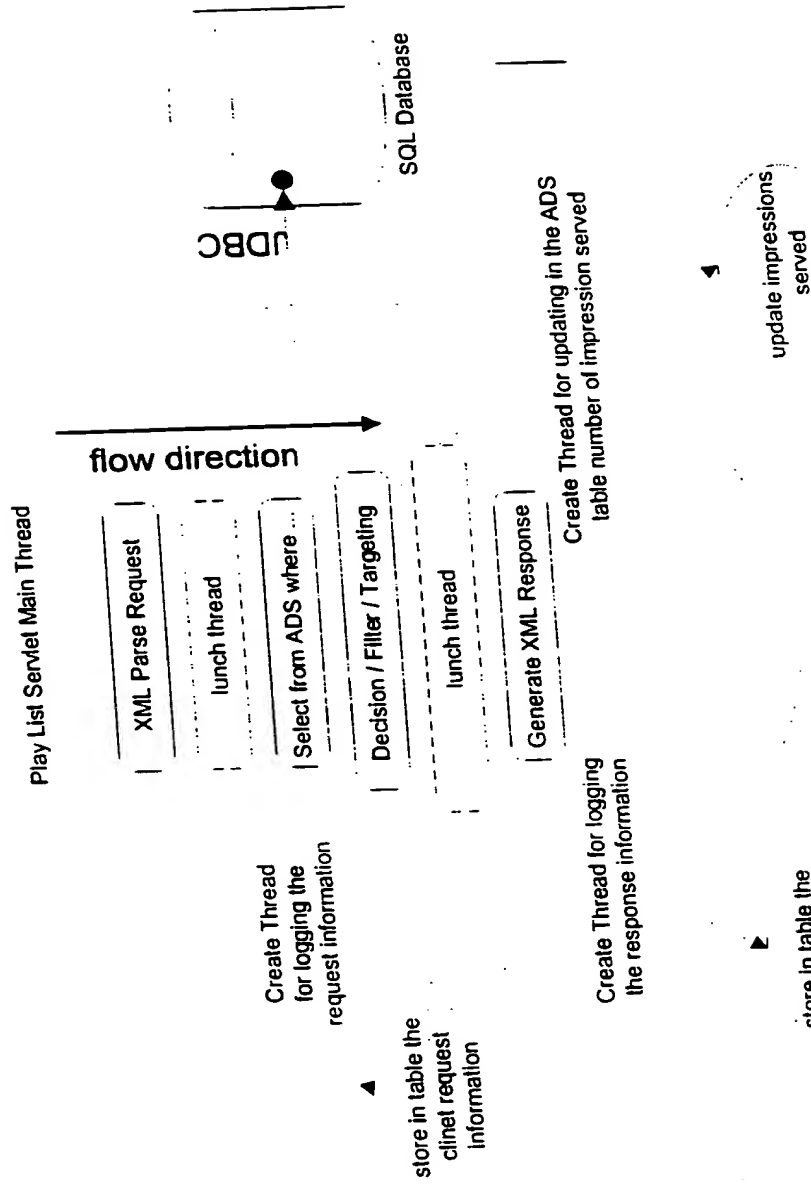


Fig. 23